ELSEVIER

Ecological Modelling 94 (1997) 33–44

ECOLOGICAL MODELLING

Designing an object-oriented structure for crop models

B. Acock*, V.R. Reddy

USDA:ARS Remote Sensing and Modeling Laboratory, Building 007, BARC-West, Beltsville, MD 20705-2350, USA

Abstract

Object-oriented design (OOD) and programming (OOP) offer many advantages for developing modular crop models. The model structure is well-defined, reuse of code is facilitated through inheritance, and data can be hidden (encapsulated) inside objects that correspond to physical components of the real system, e.g. roots, stems, leaves, or soil layers. However, OOD is best suited to describing the relationship between freely interacting objects, and it has so far been used almost exclusively for modeling simple, discrete and sequential actions. Plant models are not like the automatic teller machine software that is often used in examples of OOD. Plant organs, i.e. objects on the plant, do not wait passively for input from other organs, but they all grow in response to their environment and interact with each other simultaneously and continuously. Also, our ignorance of the processes controlling plant growth forces us to use devices like the limiting factor model to handle these interactions. Many plant models therefore calculate potential growth, limitations imposed by various factors, and then actual growth. In short, there are procedural elements in plant models that do not easily fit an OOD. However, some OOP languages like C++ allow mixed designs to be implemented, so we have developed a mixed, but mostly object-oriented structure that (1) contains the components familiar in extant procedural designs; (2) can be used for modeling at several levels of complexity; and (3) can be used to model any plant. The mixed procedural/object-oriented design has been implemented in C++ as a shell using dummy algorithms, and its operation verified. The problems and advantages are discussed.

Keywords: Procedural; Process-oriented; Discrete; Continuous; Sequential; Parallel

1. Introduction

Computer science is a rapidly developing field and new programming languages come so fast that it is difficult to keep up with them. It is important that plant modellers do not miss significant advances, but their problem is knowing which advances are significant. For instance, many computer languages have been touted as replacements for Fortran, but most plant and soil modeling is still being done in that venerable language.

The latest challenge is from object-oriented programming (OOP) and the new languages for implementing object-oriented designs (OOD).

* Corresponding author.

0304-3800/97/$17.00
PII S0304-3800(96)01926-6

Fortran was developed to implement procedural or process-oriented designs, where 'objects' such as leaves, roots, stems, etc., exist only as state variables and the code mimics the processes operating on those variables. OOP languages emphasize the real or imagined objects in a system, their states and the actions they are able to perform. The advocates of OOD claim that it has important advantages over process-oriented designs for developing modular crop models. To test the veracity of these claims, we developed a general object-oriented structure for crop models. This paper describes the structure of the model and some of the problems we encountered. Using C++, we developed a mixed procedural/object-oriented design that: (1) contains components familiar in extant procedural designs; (2) can be used for modeling at several levels of complexity; and (3) can be used to model any plant.

2. Object-oriented design

2.1. The advantages of OOD

In traditional procedural designs, a plant computer model is a collection of state variables (e.g. leaf nitrogen content, root dry weight, etc.) that describe the state of a plant system, plus algorithms that define how various processes change these state variables over time (e.g. nitrogen fixation, carbohydrate allocation, photosynthesis, etc.). The rates of the processes changing these state variables depend on the input or driving data (e.g. solar radiation, soil temperature, etc.) and the state of the system. The main practical problem with this design is how these state variables are managed. Either they are available to all the algorithms, e.g. in a Fortran COMMON block, or they are passed to each module, e.g. as arguments in a CALL statement. In the first instance, the state variables can be altered in any part of the program, there is no control over where changes take place, and it is frequently difficult to find all locations where the state variables are changed. In the second instance, a subset of the attributes is sent as a string, and the order of variable names in the sending and receiving arguments must match exactly. There are many opportunities for error.

Another problem with procedural approaches is that processes have no inherent structure to guide the design of the model. Algorithms are often grouped into modules describing related processes but each programmer's idea of the relationships differs from that of other programmers. Thus, ideas from one model must usually be reprogrammed from the original mathematical equations before they can be tested in another model, and merging models is a major task. Reuse of code is minimal.

In an object-oriented design, a plant model is a set of objects that each know their own state and how to change that state in response to commands called messages (Acock and Reynolds, 1997). The data describing the state of an object are known as attributes or variables, and the algorithms that change that state are known as services or methods. Attributes and methods are grouped together within each object. Objects can send messages to each other to initiate methods (Meyer, 1988; Wegner, 1990).

The physical objects in the real-world system impose a fairly obvious structure on the model. Even without advance agreement, there is a good chance that objects from various plant models will be similar and thus could be swapped and/or merged. Also, the data associated with an object are hidden within that object and are only available to the rest of the model if they are specifically made available. Thus an object-oriented model gives the programmer better control of data (Booch, 1991). Linking the data with the algorithms that act on them creates modules that are more independent of each other than would otherwise be possible. Finally, in OOD the objects can be arranged in an inheritance structure such that objects with the same or similar processes can inherit code from each other. All these attributes lead to maximum reuse of code(Cox, 1986; Wolczko, 1987; Meyer, 1988; Wegner, 1990).

2.2. Plant models and ATM machines

Despite these advantages, there are difficulties with using an object-oriented design for plant

models. Plants do not consist of independent objects waiting to respond to input. Fruits, leaves, stems and roots are all in constant communication via the transfer of materials and hormones essential for tissue synthesis. They are also all responding independently to their environment and are all constantly growing. Thus, plants are quite unlike the automatic teller machines (ATM), which are often cited as excellent examples of the OOD concept.

An ATM machine is inactive until it receives input from a customer. Depending on the input to the Keyboard object, the ATM will activate other objects to dispense cash, print a statement, etc. Some objects will not be activated in a given transaction. In the plant/soil/atmosphere system in plant models, all the objects are in ceaseless activity. The plant modeler is already used to representing these simultaneous continuous processes on a single processor that performs discrete calculations in sequence. With OOD the modeler is also faced with mapping these processes on to objects that may freely interact in any sequence. Most of the literature on OOD concentrates on modeling discrete sequential processes like the ATM, and the problem of modeling continuous parallel processes is rarely discussed.

The level of activity in a given object typically depends on the activities of several other objects. For example the rate of extension of the Stem object will depend on temperature from the Weather object, water uptake from the Root object, and carbon fixation by the Leaf_canopy object. (In this paper, an initial capital letter is used to denote objects, and an underscore links separate words in the name.) Because the interactions between objects are complex, there is no obvious sequence of action being passed from one object to another. Should the Leaf instruct the Root to grow or vice versa? The obvious answer is to have an object called Timer or Sequencer or Simulation_controller to instruct all the plant objects to grow in turn. This immediately imposes a procedural component on our OOD. Our Simulation_controller looks very much like the main program in a procedural code.

There is another fundamental difference between plants and ATMs. With ATMs we understand completely the processes that occur, whereas with plants our knowledge is incomplete. An ATM program is a complete physical description of what happens in an ATM but a plant model is, by definition and necessity, an approximation of a biological system. One simplification commonly used to deal with interactions in plant models is the Law of Limiting Factors (Blackman, 1905). This assumes that, when several factors are required for a process, the factor that is most limiting will determine the rate of the process (Acock et al., 1985). For example, growth depends on temperature and on supplies of carbon, water and nutrients. Many plant models calculate a potential growth rate for the plant using the factor considered most limiting and assuming that no other factors are limiting. This potential growth rate is then decremented or limited in some way for each of the other relevant factors to determine actual growth rate (Joyce and Kickert, 1987). To use this simplification, we must calculate potential growth rates, decrements or limitations and then actual growth rate. Again, we have introduced a procedural component into our OOD.

With these procedural elements in plant models, the purist tells us that we do not have an OOD and asks why we even try to use OOD. Our answer is that OOD appears to have sufficient advantages to make its use worthwhile despite the imperfections of our mixed design. Also, others have found it necessary to introduce procedural elements into OOD, but they have called them control objects (Jacobson et al., 1992) or mediators and chains of responsibility (Gamma et al., 1995). We are not convinced that these really solve the problem.

Implementing our mixed procedural/object-oriented design would be difficult in a purely object-oriented language like Smalltalk80 but C++ enables us to mix paradigms easily. The danger, of course, is that we will continue to write procedural code and miss some of the advantages of OOD.

2.3. Accommodating various levels of detail

Plant models can be built at various levels of detail, depending on their purpose. The simplest models consist of a single equation describing plant biomass as a function of time. Other models describe subcellular processes. Whereas the simplest model only requires a Plant object, other models will subdivide the Plant into Leaves, Roots, etc., and some will further subdivide Leaves into Mesophyll_cells, Xylem_vessels, and so forth. The most detailed model will not need a Plant object because the Plant will be the aggregate of all the other objects. However, there is an argument for including a Plant object.

If we start our model with a simple Plant object, we can subsequently subdivide the Plant object and pass on any information needed by the Leaf and Root objects. If we start with Leaves and Roots but no Plant, we cannot subsequently replace our complex Plant with a simple one that does not recognize the individual organs of plants. Our design has no point of attachment for a simple Plant object.

It is reasonable to ask why we would ever want to replace a detailed plant model with a simple one. Perhaps we want to use our basic design to model plants for which the quality of input data does not warrant the detail, or we want to examine species interactions in an ecosystem and need to reduce model run time. In other words, reuse of code is facilitated by developing an OOD that includes objects corresponding to higher levels of aggregation than that at which the model is written.

2.4. Processes as objects

The objects in an OOD include both attributes and algorithms to compute changes in attributes. Thus photosynthesis is a process that is performed by the Leaf_canopy object and is included in that object. Is there ever a case for making processes into separate objects? Computer scientists have recognized instances where some processes should be implemented as objects (Halbert and O'Brien, 1987; Johnson and Foote, 1988). In plant models, there are compelling reasons for treating photosynthesis and similar processes as separate objects. Photosynthesis has been studied in great detail and there are many models of the process. Hence, making this process a separate object facilitates testing and comparing the various alternative models (see examples in Chen and Reynolds, 1997 and Lemmon and Chuk, 1997).

2.5. Inheritance structure and control structure

In our mixed procedural/object-oriented design, there are two structures that must be considered: inheritance and control. In an inheritance structure the objects are arranged in a branched hierarchy such that objects at a low level are examples of the object immediately above them (Appendix A). This is sometimes called a 'kind-of' hierarchy (i.e. Mainstem is a kind of Stem, Stem is a kind of Shoot_organ, Appendix A). The inheritance structure is developed to allow several objects to use the same algorithms (Cox, 1986; Booch, 1991). For instance, if the same algorithm can be used to describe growth in Stem and Petiole, it can be placed in Shoot_organ and inherited from there by Stem and Petiole, and by all objects lower in the hierarchy. This may be the only reason for having a Shoot_organ object. In our experience, inheritance should be used with great care. It is unwise to develop inheritance structures more than about two layers deep, because debugging the code becomes very difficult. For example, if some algorithm that is used in Mainstem is inherited from System_entity via Plant_part, Shoot_organ and Stem, then its origin must be traced through each of these layers. The level of frustration rises with each step it takes to discover the algorithm. Lorenz (1993) recommended using no more than 6 levels of inheritance because of this problem, and the disadvantages of inheritance are discussed at length by Taenzer et al. (1989).

Attributes can also be inherited from superior objects in the hierarchy. Since the names will be the same in each object that inherits them, this facility should be reserved for variables that are used exclusively inside the objects. For instance, all objects below Shoot_organ in the hierarchy inherit the attribute dry_weight. If we obtain dry weights of several objects by sending them the

message.give_dry_weight, we will have to rename each dry weight to something more specific to avoid confusion.

In our control structure, the objects and messages sent to them are arranged in a call order dictated by the procedural components of the design (Appendix B). A pure OOD would not need a control structure. The calls are further arranged in a branched hierarchy such that phenomena are considered in greater detail as we descend the hierarchy (Chandy and Misra, 1988; Lamport, 1984). Thus, as shown in Appendix B, the simplest model possible would consist of only the first two levels of the hierarchy. The plant would be represented by the single object Plant, and in response to the message.grow, it would update its state. A model at the next level of detail would have Plant internally implement method.develop, method.potential growth, etc.

The model does not have to be equally developed in all the branches of the control structure. It is possible to have great detail for photosynthesis but treat water stress superficially. This may be desired in a model of a greenhouse crop where the crop is adequately watered but the use of carbon dioxide enrichment necessitates a detailed treatment of photosynthesis.

The control structure is also developed to manage data flow. Information about objects low in the hierarchy is passed through the objects above them. Thus, information about photosynthetic rate is passed to Plant through Crop_canopy. This method of handling data is not absolutely necessary. The Plant could go directly to Photosynthesis for the same information. However, in order to accommodate various levels of detail, as discussed above, it is desirable to consider data handling in the control structure.

Both the inheritance structure and the control structure need to be developed at the start of an OOD in order to determine objects that should be included in the model.

3. A proposed OOD for crop models

Our proposed OOD for crop plant models is defined by the inheritance structure in Appendix A, the control structure in Appendix B, and the listing of objects, their attributes and methods in Appendix C. It is based on an inheritance structure developed collaboratively by a group of Agricultural Research Service and university scientists who have worked together on a cotton model for several years.

Some of the objects in the inheritance structure do not appear in the control structure, e.g. Aerial_environment and Shoot_organ. This is because they are used to pass on attributes and methods to inferior objects but do not themselves participate in the storage of attributes or calculations involved in updating attributes. No part of the hierarchy is more than two layers deep and using the inheritance structure with the list of objects, their attributes and methods, it is fairly simple to locate the code of algorithms inherited from superior objects.

Near the bottom of the control structure are several messages that carry arguments. The format is: Object_name.message_to_object: argument_one:argument_two. Since this model concentrates on the plant, the Soil_environment has been left as a single, undivided object in the control structure. We have had some vigorous debates over whether this object can be usefully subdivided. At first we thought that the matrix algebra used to move materials about a two-dimensional soil profile dictated using a procedural design within the one large object. More recently we have considered making individual nodes and elements in the soil profile into separate objects.

The list of objects, their attributes, and methods is undoubtedly incomplete. In preparing it, our chief concern was to list attributes of objects that would be needed by other objects to perform their methods. There will be many more attributes needed internally in each object, but these can be left to the discretion of individual programmers. Indeed our aim has been to construct a design that leaves maximum freedom in implementation but would ensure compatibility between objects written by different programmers.

The mathematical equations used in algorithms look much the same in any computer language. The differences lie mainly in the commands that are available. Rather than having all plant model-

ers learn an object-oriented language, we can implement the proposed design as a shell using dummy algorithms to ensure that the objects interact as intended. Then modelers can replace these dummy algorithms with algorithms of their own choosing. In this way, most modelers need only learn how to write equations in the new language. They will not be concerned with how the objects interact, only that certain objects obtain attributes from other objects.

The proposed design has been implemented as a shell in C++ and the source code is available from the authors.

4. Conclusions

Object-oriented design (OOD) and programming (OOP) have thus far been used almost exclusively for modeling interactions between objects, where the actions are discrete and sequential. Interactions between objects in the plant, soil and atmosphere are complex, continuous and parallel. It is almost impossible to use a pure OOD to model plants; procedural elements have to be introduced. Putting the procedural elements into control objects does not fundamentally alter the fact that the model design is mixed. However, some OOP languages like C++ allow mixed designs to be implemented. A mixed, but mostly object-oriented structure has been developed that (1) contains the components familiar in extant procedural designs, (2) can be used for modeling at several levels of complexity and (3) can be used to model any plant.

The mixed procedural/object-oriented design has been implemented in C++ as a shell using dummy algorithms, and its operation verified. The advantages claimed for OOD are therefore available to plant modelers through the use of a mixed design. By having a computer scientist develop a shell of the model, it is possible for plant modelers to work on the algorithms in each object without learning the more complex aspects of the OOP language.

Acknowledgements

We acknowledge the programming skill of Geetha Reddy in implementing our design as a shell in C++ and testing the program. Roger Whitney gave us an especially helpful review, sharpened our thinking about the problems we discuss, and pointed out several references in the computer science literature that we had not previously encountered. He disagrees with our conclusion about the need for a mixed design and believes that the use of control objects, mediators and chains of responsibility would keep us within the OOD paradigm.

Appendix A

Proposed inheritance structure for object-oriented plant models. The only fertilizer element shown is N; other elements would be handled similarly

- Simulation_controller
- Time
- Aerial_environment
  - Two_meter_environment
  - Canopy_environment
- Crop_canopy
- Plant
- Shoot_organ
  - Stem
    - Mainstem
    - Branch_stem
  - Leaf_blade
    - Mainstem_leaf_blade
    - Branch_leaf_blade
  - Petiole
  - Internode
  - Fruiting_point
- Root_profile
- Soil_environment
- Plant_process
  - Development
  - Photosynthesis
  - Photorespiration
  - Maintenance_respiration
  - Potential_transpiration

N_acquisition
N_partitioning
C_partitioning
Management_action

Appendix B

Proposed control structure for object-oriented plant models

Simulation_controller.run
Time.step
if a new day: Two_meter_environment.interpolate
Canopy_environment.update
Plant.grow
Plant.develop
Development.update
Plant.potential_growth
Mainstem.potential_growth
Branch_stem.potential_growth
Mainstem_leaf_blade.potential_growth
Branch_leaf_blade.potential_growth
Petiole.potential_growth
Internode.potential_growth
Fruiting_point.potential_growth
Root_profile.potential_growth
Plant.water_limitations
Crop_canopy.transpire
Potential_transpiration.calculate
Plant.water_stress
Plant.N-limitations
Plant.acquire_N
N_acquisition.calculate
Plant.partition_N
N_partitioning.calculate
Plant.N_stress
Plant.C_limitations
Plant.acquire_C
Crop_canopy.photosynthesize
Photosynthesis.calculate
Crop_canopy.photorespire
Photorespiration.calculate
Crop_canopy.maintenance_respire
Maintenance_respiration.calculate
Plant.partition_C
C_partitioning.calculate
Plant.C_stress
Plant.actual_growth
Mainstem.actual_growth
Branch_stem.actual_growth
Mainstem_leaf_blade.actual_growth
Branch_leaf_blade.actual_growth
Petiole.actual_growth
Internode.actual_growth
Fruiting_point.actual_growth
Root_profile.actual_growth
Plant.abscission
Mainstem_leaf_blade.abscission
Branch_leaf_blade.abscission
Petiole.abscission
Fruiting_point.abscission
Root_profile.abscission
Management.act_if_time
Crop_canopy.set_canopy_chemical:
a_chemical to:an_amount
Soil_environment.set_chemical:
a_chemical at:node to:an_amount
Soil_environment.set_irrigation at:node
to:an_amount
Soil_environment.set_bulk_density
at:node to:an_amount
Soil_environment.update

Appendix C

Objects, their attributes and services for proposed object-oriented plant models. The values of most attributes can be obtained with the message.give_(name of attribute)

Simulation_controller
attributes or variables:
day_to_start_run
day_to_stop_run
step_size

services or methods:
.run starts the simulator

Time
attributes or variables:

date
day_of_year
hour

services or methods:
.step–move time forward one timestep
.date_from_day_of_year
.day_of_year_from_date

Aerial_environment
attributes or variables:
latitude
daylength
dawn
dusk
CO_2

services or methods:

Two_meter_environment
attributes or variables:
atmospheric_transmission_coefficient
cloud_cover_factor
for each day:
solar_radiation_integral
air temperature_max
air_temperature_min
rain_total
wind_run
water_vapor_pressure
for each time step of the day:
solar altitude
solar_azimuth
cloud_cover
diffuse/total_radiation
solar_radiation
air_temperature
rain
wind
vpd

services or methods:
.interpolate–use daily values to calculate values for each timestep

Canopy_environment
attributes or variables:
canopy_temperature
rain_intercepted
rain_on_soil

services or methods:
.update–calculate canopy environment at current time
.give_avg_temp_from: start_time to: end_time

Crop_canopy
attributes or variables:
potential_transpiration_rate
actual_transpiration_rate
canopy_albedo
leaf_transmission_coefficient
canopy_CO_2_conductance
canopy_extinction_coefficient
canopy_light_utilization_efficiency
photosynthesis_rate
photorespiration_rate
maintenance_respiration_rate
plant_population_density
leaf_area_index
leaf/canopy_area
light_interception
canopy_water_potential
canopy_osmotic_potential
canopy_chemical
canopy_chemical_amount

services or methods:
.transpire
.photosynthesize
.photorespire
.maintenance_respire
.set_canopy_chemical:a_chemical to:an_amount

Plant
attributes or variables:
cultivar
age
development_stage
potential_growth_in_dry_weight
leaf_water_potential
leaf_osmotic_potential
leaf_turgor_pressure
N_demand
C_demand
N_supply

C_supply
N_supply/demand
C_supply/demand
shoot_growing time
N_pool
C_pool

services or methods:
.grow–grow plant for one timestep
.develop
.potential_growth
.water_limitations
.water_stress
.N_limitations
.acquire_N
.partition_N
.N_stress
.C_limitations
.acquire_C
.partition_C
.C_stress
.actual_growth
.abscision

Shoot_organ
attributes or variables:
on_stem_number
location_on_stem
dry_weight
proportion_present
N_content
initiation_day
abscission_day
age
development_stage
growing
C_demand
N_demand
C_supply/demand
N_supply/demand

services or methods:

Stem
attributes or variables:
length
number_leaves_on_stem

services or methods:

Mainstem
attributes or variables:
number_of_branches

services or methods:
.potential_growth
.actual_growth
.abscission

Branch_stem
attributes or variables:

services or methods:
.potential_growth
.actual_growth
.abscission

Leaf_blade
attributes or variables:
area
thickness
water_potential

services or methods:

Mainstem_leaf_blade
attributes or variables:

services or methods:
.potential_growth
.actual_growth
.abscission

Branch_leaf_blade
attributes or variables:

services or methods:
.potential_growth
.actual growth
.abscission

Petiole
attributes or variables:
length

services or methods:
.potential_growth
.actual_growth
.abscission

Internode
attributes or variables:
length

services or methods:
.potential_growth
.actual_growth
.abscission

Fruiting_point
attributes or variables:
stage_of_development
fruit_dry_weight
potential_seed_dry_weight
actual_seed_dry_weight

services or methods:
.potential_growth
.actual_growth
.abscission

Root_profile
attributes or variables:
for each node in the soil profile:
young_root_length
old_root_length
young_root_dry_weight
old_root_dry_weight
root_water_uptake
root_N03_uptake
root_P_uptake
root_K_uptake

services or methods:
.water_uptake
.N03_uptake
.potential_growth
.actual_growth
.abscision

Soil_environment
attributes or variables:
for each node in the soil profile:
sand
silt
clay
bulk_density
total_pore_space
hydraulic_conductivity
soil_water_content
soil_water_potential
soil_osmotic_potential
soil_temperature
soil_N
soil_N03
soil_NH4
soil_P
soil_K
soil_Cl
soil_pH
soil_O2
soil_CEC
soil_organic_matter
for upper boundary nodes:
runoff
soil_water_evaporation
for lower boundary nodes:
deep_drainage

services or methods:
.update-calculate soil environment at current time
.set chemical:a_chemical at:node to:an_amount
.set_irrigation at:node to:an_amount
.set_bulk_density at:node to:an_amount

Plant_process
attributes or variables:

services or methods:

Development
attributes or variables:
development_stage
change_in_development_stage

services or methods:
.update

Photosynthesis
attributes or variables:
photosynthesis_rate

services or methods:
.calculate

Photorespiration
attributes or variables:
photorespiration_rate

services or methods:
.calculate

Maintenance_respiration
attributes or variables:
maintenance_respiration_rate

services or methods:
.calculate

Potential_Transpiration
attributes or variables:
potential_transpiration_rate

services or methods:
.calculate

N_acquisition
attributes or variables:
N_supply

services or methods:
.calculate

N_partitioning
attributes or variables:
for each shoot organ:
N_supply/demand

services or methods:
.calculate

C_partitioning
attributes or variables:
for each shoot organ:
C_supply/demand

services or methods:
.calculate

Management
attributes or variables:
row_orientation
row_pattern
row_spacing
sowing_day
sowing_depth
seed_density
emergence_day
emerged_plant_density
target_maturity_day
day_of_action
type_of_action
irrigation_location
irrigation_rate
irrigation_duration
irrigation_chemical_type
irrigation_chemical_conc
chemical_applied
chemical_application_location
chemical_application_amount
tillage_location
tillage_depth

services or methods:
.act_if_time-check to see if a management action takes place at this time

References

Acock, B., Reddy, V.R., Whisler, F.D. et al., 1985. The soybean crop simulator GLYCIM: model documentation. No. PB85 171163-AS, USDA, Washington, DC.

Acock, B. and Reynolds, J.F., 1997. Introduction: modularity in plant growth models. Ecol. Model., 94: 1–6.

Blackman, F.F., 1905. Optima and limiting factors. Ann. Bot. (old series), 19: 281-295.

Booch, G., 1991. Object-oriented design with applications. Benjamin-Cummins, Redwood City, CA, 580 pp.

Chandy, K.M. and Misra, J., 1988. Parallel Program Design: A Foundation. Addision-Wesley, Reading, MA, 512 pp.

Chen, J.-L. and Reynolds, J.F., 1997. GePSi: a generic, plant growth simulator based on object-oriented principles. Ecol. Model., 94: 53–66.

Cox, B., 1986. Object-oriented programming: an evolutionary approach. Addison-Wesley, Reading, MA.

Gamma, E., Helm, R.R.J. and Vlissides, J., 1995. Design patterns: elements of reusable object-oriented software. Addison-Wesley, Reading, MA, 395 pp.

Halbert, D.C. and O'Brien, P.D., 1987. Using types and inheritance in object-oriented programming. IEEE Software, 4: 71–79.

Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G., 1992. Object-oriented software engineering: a use case driven approach. Addison-Wesley, Reading, MA, 524 pp.

Johnson, R.E. and Foote, B., 1988. Designing reusable classes. J. Object-Oriented Programming, June/July: 2 -35.

Joyce, L.A. and Kickert, R.N., 1987. Applied plant growth models for grazinglands, forests and crops. In: K. Wisiol and J.D. Hesketh (Editors), Plant Growth Modeling for Resource Management. CRC Press, Boca Raton, FL.

Lamport, L., 1984. Solved problems, unsolved problems, and non-problems in concurrency. 3rd Annual ACM Symp. Principles of Distributed Computing, Montreal, Canada.

Lemmon, H. and Chuk, N., 1997. Object-oriented design of a cotton crop model. Ecol. Model., 94: 45 -51.

Lorenz, M., 1993. Object-oriented software development: a practical guide. Prentice-Hall, Englewood Cliffs, NJ, 227 pp.

Meyer, B., 1988. Object-oriented software construction. Prentice Hall, Englewood Cliffs, NJ, 534 pp.

Taenzer, D., Ganti, M. and Podar, S., 1989. Object-oriented software reuse: the yoyo problem. J. Object-Oriented Programming, 2: 30–35.

Wegner, P., 1990. Concepts and paradigms of object-oriented programming. Oops Messenger, 1: 7–87.

Wolczko, M., 1987. Semantics of Smalltalk-80. Eur. Conf. Proc. Object Oriented Programming. Springer-Verlag, Paris, France, pp. 108–120.